Programare avansata pe obiecte – laborator 5

***Andreea Ciocan*** *Andreea.ciocan@endava.com* ***https://github.com/deeas/PAO-2024***

# Aggregation & Composition

In programarea orientate obiect, un obiect comunica cu alte obiecte pentru a folosi funcitonalitea si serviciile acelor obiecte. Composition and Aggregation sunt doua forme de asociere

**Aggregation**

* Reprezinta o relatie de tipul Has-A
* Este o asociere unidirectionala, ex: un department poate avea student
* Ambele entitati pot supravietui individual, insemnand ca distrugand o entitate nu va afecta cealalta entitate

**Composition**

* Reprezinta o asociere de tipul part-of
* In compositie, ambele entitati depend una de alta
* Clasa continuta nu poate exist afara container. Daca containerul este distrus, copilul este si el distrus

# Records

* Clase imutabile utilizate pentru a manipula o multimer fixa de valori
* Nu pot fi abstracte
* Nu pot fi extinse
* Nu pot extinde alte clase
* Pot implementa una sau mai multe interfete

# Clase abstracte si interfete

Folosim o **clasa abstracta** atunci cand vrem sa:

* Implementam doar unele metode din clasa
* Reutilizam o serie de metode si membri din aceasta clasa in clasele derivate
* Nu vrem sa instantiem clasa

Particularitati:

* Putem avea metode/date membru cu **orice** modificator/non modificator de acces

Folosim **interfete** atunci cand vrem sa:

* Avem o descriere a structurii fara implementari

o Metodele sunt implicit public

* Definim un contract intre clase

Particularitati interfete:

* Putem crea folosind cuvantul cheie: **interface**
* Pentru a defini o clasa conforma cu o interfata folosim cuvantul cheie **implements**
* Pentru a defini o interfata care mosteneste alta interfata folosim cuvantul cheie **extends**
* Putem avea **campuri**, dar acestea sunt in mod implicit **static** si **final**
* Combinarea unor interfete care contin o metoda cu acelasi nume e posibila doar daca metodele nu au tipuri intoarse diferite si aceeasi lista de argumente. Este preferabil ca in interfete care trebuie combinate sa nu existe metode cu acelasi nume, pentru a evita confuziile.
* Inainte sa folosim o interfata ne trebuie o clasa care sa o implementeze, ele **nu pot fi instantiate**
* **Putem avea** **metode** fara implementare (public si abstract), default (vizibilitate public, apar din java 8), statice (vizibilitate public, apar din java 8), cu vizibilitate private (din java 9, statice sau nestatice)

A screenshot of a screen

Description automatically generated

# Comparator si Comparable

* Interfete folosite pentru sortare
* Pentru a folosi **Comparable** clasa trebuie sa implementeze aceasta interfata, fiecare clasa care face asta putand define **un criteriu** de sortare.
* Pentru a folosi **Comparator**, cream o clasa separata care implementeaza interfata si prin urmare metoda compare in care definim criteriul de sortare dorit. Folosind aceasta abordare clasele noastre pot defini **mai multe criterii** de sortare.

Predict the output

class Base {  
 public void show() {  
 System.*out*.println("Base::show() called");  
 }  
}  
  
class Derived extends Base {  
 public void show() {  
 System.*out*.println("Derived::show() called");  
 }  
}  
  
class Main {  
 public static void main(String[] args) {  
 Base b = new Derived();  
 b.show();  
 }  
}

class Base {

final public void show() {

System.out.println("Base::show() called");

}

}

class Derived extends Base {

public void show() {

System.out.println("Derived::show() called");

}

}

class Main {

public static void main(String[] args) {

Base b = new Derived();

b.show();

}

}

class Base {

public static void show() {

System.out.println("Base::show() called");

}

}

class Derived extends Base {

public static void show() {

System.out.println("Derived::show() called");

}

}

class Main {

public static void main(String[] args) {

Base b = new Derived();

b.show();

}

}

class Base {

public void print() {

System.out.println("Base");

}

}

class Derived extends Base {

public void print() {

System.out.println("Derived");

}

}

class Main {

public static void doPrint(Base o) {

o.print();

}

public static void main(String[] args) {

Base x = new Base();

Base y = new Derived();

Derived z = new Derived();

doPrint(x);

doPrint(y);

doPrint(z);

}

}class Base {

public void foo() {

System.out.println("Base");

}

}

class Derived extends Base {

private void foo() {

System.out.println("Derived");

}

}

public class main {

public static void main(String args[]) {

Base b = new Derived();

b.foo();

}

}

public class Base {

private int data;

public Base() {

data = 5;

}

public int getData() {

return this.data;

}

}

class Derived extends Base {

private int data;

public Derived() {

data = 6;

}

private int getData() {

return data;

}

public static void main(String[] args) {

Derived myData = new Derived();

System.out.println(myData.getData());

}

}

public class Test {

private int data = 5;

public int getData() {

return this.data;

}

public int getData(int value) {

return (data + 1);

}

public int getData(int... value) {

return (data + 2);

}

public static void main(String[] args) {

Test temp = new Test();

System.out.println(temp.getData(7, 8, 12));

}

}

class Helper {

private int data;

private Helper() {

data = 5;

}

}

public class Test {

public static void main(String[] args) {

Helper help = new Helper();

System.out.println(help.data);

}

}

# Exercitii

1. Declarati o interfata Task care contine o metoda execute(), care returneaza void. Pe baza acestei interfete implementati 3 clase: RandomTask, OutTask si CounterOutTask.
   1. Pentru OutTask afisati un mesaj in consola, mesaj specificat n constructor
   2. Pentru RandomTask generati un numar aleator si afisati un mesaj cu el. Generarea se face in constructor
   3. Pentru CounterOutTask, incrementati un contor global si afisati-i valoarea dupa fiecare incrementare

Creati o noua clasa Container in care puteti adauga si elimina elemente.

1. Declarati o clasa Album care are campurile: nume, artist, rating si anul publicarii.
   1. Sortati un array de albume pe baza numelui, rating-ului si anului publicarii. Folositi ambele interfete de comparare.
   2. Creati o clasa Main unde declarati array-ul si afisati-l inainte si dupa sortare.
2. Creati 4 interfete Minus, Plus, Mult si Div care contin cate o metoda aferenta numelui si are ca argument un numar de tipul float. Declarati o clasa Operation care sa le implementeze si care are un camp de tip float, modificat de metodele implementate de voi.